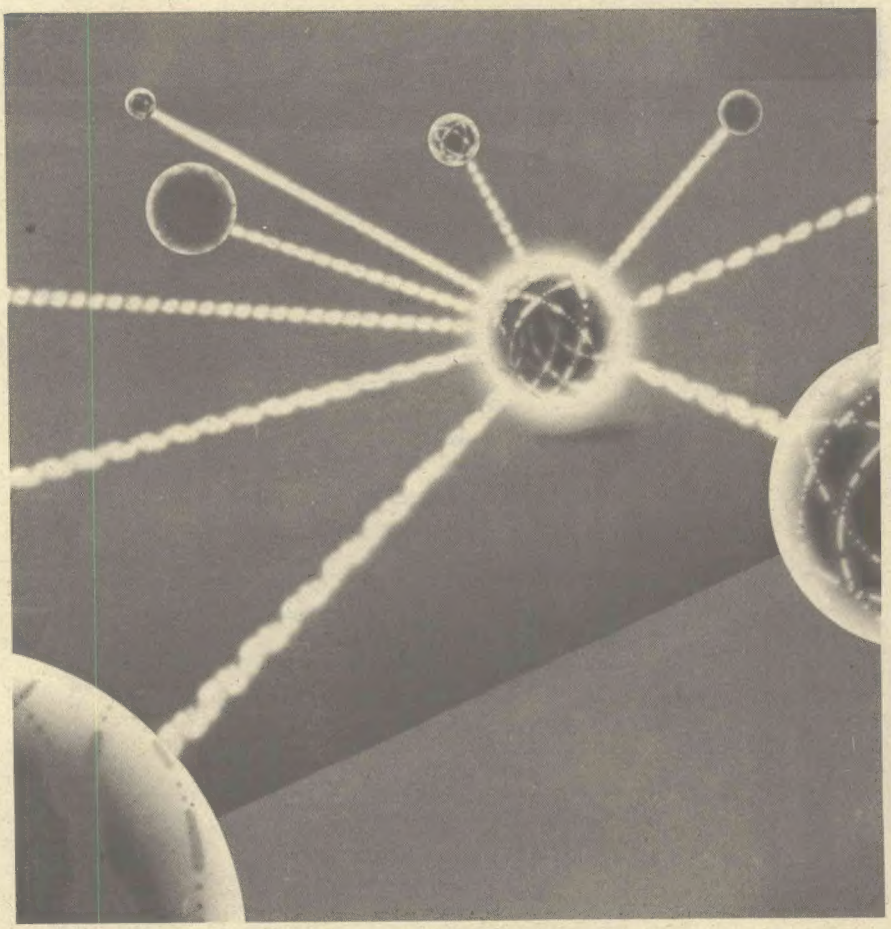
ประวิทย์ เรืองศาสตร์

พูดจาภาษา SQL ตอนที่ 3

การใช้ไวยกรณ์ของคำสั่ง SELECT ในฟอกซ์โปร 2.0 ในส่วนที่เกี่ยวข้องกับ SQL



โอาล่ะครับถึงตอนสำคัญเสียที จากตอนที่แล้วในวารสารไมโครฯ ฉบับที่ 83 ซึ่งผมได้บอกเอาไว้ว่าจะนำเสนอรายละเอียดของคำสั่ง SELECT ซึ่งถือว่าเป็นคำสั่งที่สำคัญ และรวมไปถึงเรื่องราวต่าง ๆ ที่จำเป็นจะต้องทราบในภาษา SQL ที่มีในโปรแกรมฟอกซ์โปร 2.0 (ต่อจากนี้ผมจะใช้คำว่า "ฟอกซ์-โปร" แทนนะครับ) ดังนั้นบทความในตอนนี้ก็เปรียบเสมือนบทเข้าพระเข้านางเลยทีเดียว เพราะฉะนั้นผู้อ่านที่ได้ติดตามมาในตอนก่อน ๆ นี้ ก็ไม่ควรพลาดเป็นอย่างยิ่งนะครับ แต่ก่อนที่เราจะเปิดฉากออกโรงกันต่อไป ผมขออนุญาตชี้แจงเกี่ยวกับเรื่องราวบางอย่างของฟอกซ์โปร ในส่วนที่เกี่ยวข้องกับ SQL เสียก่อน ซึ่งผมเห็นว่าถ้าหากท่านผู้อ่านได้ทราบและมีความเข้าใจที่ถูกต้องเกี่ยวกับ SQL ในฟอกซ์โปร ก็จะเป็นประโยชน์ต่อไปในภายภาคหน้า ถ้าหากท่านได้มีโอกาสไปใช้ภาษา SQL ของซอฟต์แวร์ตัวอื่น ๆ

ความแตกต่าง

เนื่องจากบรรดาโปรแกรมตระกูล Xbase ทั้งหลาย (dBASE, FOXBase, FoxPro, และอื่น ๆ) ได้ถูกสร้างให้มีการวางรูปแบบการจัดการฐานข้อมูล ในลักษณะคล้ายคลึงกับระบบการจัดการฐานข้อมูลสัมพันธ์ (Relational Database Management System ; RDBMS) ซึ่งถ้าหากเราไม่เคร่งครัดกับกฎเกณฑ์จนเกินไปก็พอกล่าวได้ว่า โปรแกรมตระกูล Xbase เหล่านี้เป็นโปรแกรมที่จัดอยู่ในประเภท RDBMS แต่ถ้าหากเรามองย้อนไปถึงบทบัญญัติ 12 ประการ ที่กล่าวถึงคุณสมบัติของฐานข้อมูลสัมพันธ์ (เป็นที่รู้จักกันในชื่อ "Rule Zero") ของ E.F. Codd ซึ่งถือได้ว่าเป็นปรมาจารย์ผู้ให้กำเนิด RDBMS (ซึ่งผมไม่ได้นำมาแสดงไว้ในบทความนี้เพราะยาวมาก) ก็จะพบว่ามีอยู่หลายข้อ ที่โปรแกรม Xbase เหล่านี้ยังไม่สามารถปฏิบัติตามได้ (มาก

กว่า 6 ข้อ) ดังนั้นจึงไม่สามารถกล่าวได้เต็มปากเต็มคำนักว่า โปรแกรม Xbase เหล่านี้เป็น RDBMS อย่างสมบูรณ์แบบ และที่ผมเอาเรื่องนี้มากล่าว ก็ไม่ได้มีเจตนาที่จะใส่ร้ายป้ายสีหรือว่าจับผิดโปรแกรมเหล่านี้แต่อย่างใด เพียงแต่ว่าผมพยายามชี้ให้เห็นถึงความแตกต่างเบื้องต้น ซึ่งจะนำไปสู่ความแตกต่างอันถัดไปที่ปรากฏในภาษา SQL ในฟอกซ์โปร

ความจริงแล้วภาษา SQL ที่มีในฟอกซ์โปร ก็ได้พัฒนาตามแนวทางมาตรฐานของ ANSI อยู่แล้วเพียงแต่ว่ามีบางอย่างที่ต้องแก้ไข เนื่องด้วยความจำเป็นบางประการ โดยเรื่องที่ได้แก้ไขไปจากมาตรฐานมีดังต่อไปนี้

1. เพื่อปรับให้เข้ากับรูปแบบขององค์ประกอบเดิมในฟอกซ์โปร ซึ่งได้แก่ เรื่องต่อไปนี้ ประเภทของข้อมูล, การตั้งชื่อ, การใส่เครื่องหมายวรรคตอน ซึ่งยังยึดหลักเกณฑ์เดิมที่มีอยู่แล้วในฟอกซ์โปร

2. จากที่ผมได้เรียนให้ทราบไว้ข้างต้นแล้วว่าฟอกซ์โปรไม่ได้เป็น RDBMS สมบูรณ์แบบ ดังนั้นจึงมีบางอย่างที่เกี่ยวข้องกับกฎเกณฑ์ของ RDBMS ที่ SQL ในฟอกซ์โปรไม่สามารถตอบสนองได้เช่น ฟอกซ์โปรไม่มีข้อมูลชนิด NULL และเรื่องของ primary key (ผมได้อธิบายความของข้อมูลแบบ NULL ไว้ในรูปที่ 1 ด้วยครับ)

3. เป็นเรื่องของการพบกันครึ่งทางที่เหมาะเจาะที่สุดระหว่าง SQL กับฟอกซ์โปร (ซึ่งเก่งคนละอย่าง) เพราะว่าโดยปกติแล้วภาษา SQL เป็นภาษาที่มีรูปแบบการจัดการเข้าถึงข้อมูล ในลักษณะกลุ่มของข้อมูล (เซต) ซึ่งแต่ก่อนนี้ภาษา SQL ถูกใช้ในลักษณะเป็นส่วนเสริมของภาษาคอมพิวเตอร์ทั่วไป ที่ไม่มีฟังก์ชันโดยตรง ที่ใช้ในการจัดการกับฐานข้อมูลสัมพันธ์ (ยกตัวอย่างเช่น ภาษาซี, โคบอล) ก็เลยทำให้เกิดปัญหาอยู่พอสมควร ในการส่งผ่านค่าตัวแปรระหว่างภาษาที่ใช้กับคำสั่ง SQL แต่สำหรับฟอกซ์โปรซึ่งเป็นภาษาที่จัดการและเข้าถึงข้อมูล ได้ในลักษณะของเรคอร์ดได้เป็นอย่างดีอยู่แล้ว และยังได้ SQL มาช่วยอีกแรง โดยคำสั่งของ SQL ในฟอกซ์โปรถูกวางตัวอยู่ในระดับเดียวกันกับคำสั่งของฟอกซ์โปร (ไม่ต้องมีกรรมวิธีการผ่านค่า) ทำให้ทั้งสองอย่างก็รวมกันได้อย่างค่อนข้างสนิทชิดเชื้อ ก็เลยยิ่งทำให้เก่งระเบิดไปเลยนะซิครับ และในจุดนี้ก็เป็นการลบปมด้อยของภาษา SQL ในข้อที่บอกว่า SQL เหมาะกับการประมวลผลแบบแบตช์เท่านั้น ซึ่งผมได้นำมากล่าวไว้ในตอนที่แล้ว ไปโดยสิ้นเชิง

> NULL values หมายถึง ชนิดของข้อมูลที่กำหนดขึ้นใช้ใน RDBMS โดยถือว่าค่าของ NULL ไม่ได้มีค่าเป็นอย่างหนึ่งอย่างใด เช่น 0, Blank หรืออื่นๆ และค่าของ NULL ในฟิลด์ข้อมูลต่างชนิดกันก็อาจจะมีค่าไม่เท่ากันก็ได้ ในภาษา SQL ที่มีการใช้ค่า NULL ก็จะใช้เพื่อแสดงถึง ความไม่สมบูรณ์ของข้อมูลในฟิลด์นั้นๆ (ซึ่งอาจให้ความหมายได้ว่า ยังไม่มีการใส่ข้อมูลลงในฟิลด์นั้นหรือข้อมูลที่พยายามบรรจุใส่ในฟิลด์นั้นเป็นค่าที่ยอมรับไม่ได้)
>
> ในการใช้งานแฟ้มข้อมูลระบบ RDBMS บางครั้งอาจจะอนุญาตให้ข้อมูลในบางฟิลด์ สามารถเป็นค่า NULL ได้ (ในกรณีที่ข้อมูลในฟิลด์นั้นไม่มีความสำคัญหรือจำเป็นต่อระบบมากนัก) แต่ถ้าหากเป็นข้อมูลที่สำคัญ ก็สามารถป้องกันไม่ให้เกิด NULL values ขึ้นในฟิลด์นั้นๆ ได้ โดยการกำหนดคีย์เวิร์ด NOT NULL ให้กับฟิลด์นั้นเมื่อทำการสร้างแฟ้มข้อมูล ตัวอย่างเช่น ฟิลด์รหัสประจำตัว เป็นต้น ทั้งนี้เพราะถ้าหากฟิลด์รหัสประจำตัวเกิดเป็น NULL ขึ้นมาแล้ว ก็จะทำให้ข้อมูลที่ถูกเก็บเอาไว้ไม่สามารถอ้างถึงได้ (หรือทำได้โดยยาก) ดังนั้นใน RDMBS ที่เต็มรูปแบบ จึงไม่ยอมจัดเก็บข้อมูลของเรคอร์ดนั้นๆ ถ้าหากว่าฟิลด์ที่มีการระบุ NOT NULL ยังมีค่าเป็น NULL อยู่

รูปที่ 1 ความหมายของข้อมูลชนิด NULL

เรียบเรียงแนวคิด

ผมขอย้อนหลังสักนิดนะครับ อันดับแรกผมอยากให้ท่านดู โครงสร้างคำสั่ง SELECT ที่ผมได้แสดงไว้ในบทความตอนที่ 1 ในไมโครฯ ฉบับที่ 80 ประกอบไปด้วยก็จะเป็นการดี และจากรูปที่ 2(ก) ซึ่งผมได้แสดงรูปแบบการใช้งานพื้นฐานของคำสั่ง SELECT

```
SELECT * ;
FROM customer ;
WHERE customer.state = "NY"
```

ตัวอย่างที่ 2 (ก)

```
SELECT  CUS.company, CUS.city, CUS.state ;
FROM    customer CUS, offices ;
WHERE   customer.city = offices.city ;
        AND customer.state = offices.state
```

ตัวอย่างที่ 2 (ข)

รูปที่ 2 ตัวอย่างการใช้คำสั่ง SQL ในขั้นพื้นฐาน

ท่านจะเห็นว่าประกอบด้วยอนุประโยค 3 ส่วนด้วยกันคือ SELECT, FROM, WHERE ซึ่งท่านก็ได้ทราบถึงหน้าที่ของแต่ละอนุประโยคไปบ้างแล้ว (ผมได้อธิบายเอาไว้ในตอนที่ 2 ของบทความชุดนี้) ทีนี้เราก็จะมาหาแนวทาง เพื่อให้ท่านสามารถใช้คำสั่ง SELECT ได้อย่างเป็นขั้นเป็นตอนกันก่อนล่ะ

ในการใช้คำสั่ง SELECT เพื่อทำการสอบถามข้อมูล มีสิ่งแรกที่ต้องคำนึงถึงคือ แฟ้มข้อมูลที่เกี่ยวข้องในการสอบถามเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ นั่นคือ ท่านต้องเริ่มจากการจัดเตรียมรายละเอียดในอนุประโยค FROM ยกตัวอย่างเช่น สมมติว่าท่านต้องการจะหาว่า มีบริษัทลูกค้ารายใดบ้างที่อยู่ในเมือง/รัฐเดียวกันกับสำนักงานขายของบริษัท จากคำถามอันนี้จะเห็นว่าแฟ้มข้อมูลที่ต้องการก็คือ แฟ้ม CUSTOMER.DBF และ OFFICES.DBF ซึ่งผมได้แสดงตัวอย่างคำสั่งเอาไว้ในรูปที่ 2(ข) สำหรับวิธีการคิด ผมขอเสนอวิธีซึ่งทำได้ง่าย ๆ ดังนี้ครับ ให้ทดลองถามตัวท่านเองว่า ถ้าให้ท่านเป็นผู้ค้นหาผลลัพธ์ด้วยตัวเอง ท่านต้องการข้อมูลจากแฟ้มข้อมูลใดบ้าง ซึ่งก็จะทำให้ท่านได้ชื่อแฟ้มข้อมูลที่เกี่ยวข้องทั้งหมด และสามารถนำมาประกอบเป็นอนุประโยค FROM ได้แล้ว ต่อจากนั้นก็ค่อยคิดเรื่องของผลลัพธ์ที่ต้องการให้แสดงออกมา ซึ่งเป็นส่วนที่ระบุต่อท้ายในอนุประโยค SELECT โดยปกติแล้วผลลัพธ์ของการสอบถามข้อมูลก็จะเป็นชื่อฟิลด์ในแฟ้มข้อมูลที่ระบุไว้ในอนุประโยค FROM นั่นเอง (จะเป็นชื่อฟิลด์ที่ไม่มีอยู่ในแฟ้มข้อมูลไม่ได้โดยเด็ดขาดนะครับ) และนอกจากนี้แล้วท่านยังสามารถนำเอาค่าคงที่ใด ๆ มาแสดงเป็นผลลัพธ์ หรือใช้ฟังก์ชันเข้าจัดการกับข้อมูลในฟิลด์ใดฟิลด์หนึ่งเพื่อให้ปรากฏในตารางผลลัพธ์ได้อีกด้วย ผมได้แสดงตัวอย่างของการใช้ฟังก์ชันไว้ในรูปที่ 3 ส่วนในรายละเอียดในการใช้ ผมขอยกไปชี้แจงในอีกหัวข้อหนึ่งก็แล้วกันครับ

```
SELECT detail.pno, parts.descript, ;
                SUM(qty), SUM(qty*detail.price) ;
FROM            detail, parts ;
WHERE           detail.pno = parts.pno ;
GROUP BY detail.pno ;
HAVING          SUM(qty) > 50
```

รูปที่ 3 การใช้ฟังก์ชันร่วมกับฟิลด์ผลลัพธ์

ความจริงแล้ว เมื่อท่านได้รายละเอียดทั้งในส่วนของอนุประโยค SELECT และ FROM เรียบร้อยแล้ว ก็ถือได้ว่าคำสั่ง SELECT พร้อมที่จะทำงานได้แล้ว แต่ว่ายังมีส่วนของอนุประโยค WHERE ซึ่งใช้ในการกำหนดเงื่อนไข เพื่อระบุถึงลักษณะของกลุ่มข้อมูลที่ต้องการ ซึ่งในรายละเอียด ผมแยกไปกล่าวไว้ในหัวข้อ "การสร้างเงื่อนไข" และนอกจากอนุประโยคหลักทั้งสามนี้แล้วยังมีอนุประโยคอื่น ๆ ที่ใช้แต่งเติมเสริมส่งให้ผลลัพธ์ที่ได้ สมบูรณ์ยิ่งขึ้นอีกคือ GROUP BY, HAVING, ORDER BY และอื่น ๆ อีก

ดูไวยกรณ์

ความจริงแล้ว ในคู่มือฟอกซ์โปรก็ได้ให้รายละเอียดเกี่ยวกับการใช้งานเอาไว้อย่างสมบูรณ์แล้ว แต่ผมก็อยากเผื่อไว้สำหรับผู้อ่านที่ไม่มีคู่มือ จะได้รู้ไวยกรณ์ของคำสั่งอย่างละเอียด ก็เลยขอนำมากล่าวไว้ในที่นี้ด้วย เอาล่ะครับ เพื่อให้ไม่เป็นการเสียเวลา เราก็จะพิจารณาอนุประโยคต่าง ๆ ที่ประกอบอยู่ในคำสั่ง SELECT ไปเป็นลำดับ เริ่มเลยนะครับ

- SELECT โดยอนุประโยค SELECT จะใช้ระบุถึงชื่อฟิลด์ที่ต้องการให้ปรากฏในผลลัพธ์ซึ่งอาจจะเป็นชื่อฟิลด์ที่มาจากแฟ้มข้อมูลมากกว่าหนึ่งแฟ้มก็ได้ และเพื่อป้องกันข้อผิดพลาดที่อาจจะเกิดขึ้นในกรณีที่มีชื่อฟิลด์ซ้ำกันในแฟ้มข้อมูลที่ต่างกันให้ท่านใส่ชื่อแฟ้มข้อมูลนำหน้าแล้วคั่นด้วย "." แล้วจึงตามด้วยชื่อฟิลด์ (ดังที่แสดงไว้ในตัวอย่างข้างล่าง) ในบางโอกาส หากท่านไม่ต้องการระบุชื่อฟิลด์ ท่านสามารถใช้เครื่องหมาย "*" ต่อท้ายอนุประโยค SELECT เพื่อเป็นการบอกว่าให้เอาทุก ๆ ฟิลด์ปรากฏในผลลัพธ์ (เครื่องหมาย "*" มีความหมายเท่ากับ All Field) นอกจากนี้แล้ว ถ้าหากดูรูปแบบคำสั่ง ท่านจะเห็นว่าในอนุประโยค SELECT ของฟอกซ์โปรยังมีรายละเอียดเพิ่มเติมอีกคือ [ALL : DISTINCT] ต่อท้ายอยู่ ให้ท่านเลือกใช้อย่างใดอย่างหนึ่ง โดยปกติแล้วฟอกซ์โปรจะแสดงข้อมูลทุกรายการที่เข้าในเงื่อนไขของคำสั่ง SELECT ซึ่งมีค่าเท่ากับ 'SELECT ALL แต่ถ้าหากท่านต้องการตัดรายการข้อมูลที่ซ้ำกันออกไปจากตารางผลลัพธ์ก็ให้ระบุคำว่า DISTINCT ต่อท้ายอนุประโยค SELECT ด้วย และเพื่อให้ท่านเห็นความแตกต่างให้ทดลองใช้คำสั่งดังต่อไปนี้ (แล้วสังเกตดูว่าได้ผลลัพธ์ต่างกันอย่างไรบ้าง)

```
- SELECT detail.ino ;
  FROM detail
- SELECT DISTINCT detail.ino ;
  FROM detail
```

- FROM ใช้ในการระบุชื่อของแฟ้มข้อมูลที่เกี่ยวข้องกับการสอบถามข้อมูลนั้น ๆ และดูเหมือนเป็นกฎเกณฑ์ว่าเมื่อใช้คำสั่ง SELECT ทุกครั้งจะขาดอนุประโยค FROM ไปไม่ได้ สำหรับในฟอกซ์โปร การระบุชื่อแฟ้มข้อมูลในอนุประโยค FROM ยังสามารถกำหนดชื่อเทียมชั่วคราวได้อีกด้วย ทั้งนี้เพื่ออำนวยความสะดวกในการระบุชื่อฟิลด์ในส่วนกำหนดผลลัพธ์ (ในกรณีที่ผลลัพธ์ประกอบด้วยชื่อฟิลด์ของแฟ้มข้อมูลมากกว่า 1 แฟ้ม) โดยที่ชื่อเทียมที่กำหนดขึ้นชั่วคราวนี้จะไม่มีผลกระทบกับชื่อเทียม (ALIAS) ที่กำหนดไว้เมื่อตอนเปิดแฟ้มข้อมูล ผมได้แสดงตัวอย่างไว้ในรูปที่ 4

```
SELECT C.cno, C.address, In.ino ;
FROM   customer C, invoices In ;
WHERE  C.cno = In.cno
```

รูปที่ 4 การกำหนดชื่อเทียมชั่วคราวในอนุประโยค FROM

- WHERE เป็นอนุประโยคที่ใช้ระบุเงื่อนไขเพื่อค้นหาเรคอร์ดในแฟ้มข้อมูลที่ได้ระบุไว้ในอนุประโยค FROM และดึงเอาเรคอร์ดที่เข้าเงื่อนไขที่ระบุในอนุประโยค WHERE ไปเป็นผลลัพธ์ เงื่อนไขที่กำหนดต่อท้ายอนุประโยค WHERE จะแบ่งออกเป็น 2 ประเภท คือ

1. Filter condition เป็นเงื่อนไขซึ่งระบุถึงลักษณะของกลุ่มเรคอร์ดผลลัพธ์ที่ต้องการ โดยสามารถตั้งเงื่อนไขได้มากกว่า 1 เงื่อนไข โดยสามารถเชื่อมเงื่อนไขแต่ละอันเข้าด้วยกันด้วยตัวกระทำการ AND, OR, NOT

2. Join condition เป็นเงื่อนไขที่ใช้ในการเชื่อมโยงระหว่างแฟ้มข้อมูล โดยในกรณีที่มีการสอบถามโดยใช้แฟ้มข้อมูลตั้งแต่ 2 แฟ้มขึ้นไป ต้องมีการกำหนดเงื่อนไขสำหรับการเชื่อมโยงไม่เช่นนั้นแล้วอาจจะเกิดผลลัพธ์ที่ไม่พึงประสงค์ก็ได้ (ในคู่มือฟอกซ์โปรก็ได้เตือนให้ระวังในจุดนี้เอาไว้ด้วย) ในการกำหนดเงื่อนไขในการเชื่อมโยงมีหลักเกณฑ์อย่างง่าย ๆ คือ ใช้ฟิลด์ที่มีความหมายเหมือนกันในแต่ละแฟ้มข้อมูลมาเป็นตัวเชื่อมโยง โดยไม่ต้องพะวงว่าฟิลด์เหล่านั้นจะต้องเป็นคีย์ดัชนีหรือไม่ (แต่ถ้าเป็นก็จะช่วยเพิ่มความเร็วในการเชื่อมโยง) เพราะว่า SQL จะทำการสร้างดัชนีชั่วคราวขึ้นใช้ระหว่างทำการสอบถามโดยอัตโนมัติอยู่แล้ว

- GROUP BY เป็นอนุประโยคที่วางตัวอยู่ต่อจากอนุประโยค WHERE แต่ถ้าหากไม่มีการระบุอนุประโยค WHERE ในคำสั่ง ก็สามารถใช้ GROUP BY ต่อจากอนุประโยค FROM ได้เลย โดยอนุประโยค GROUP BY จะใช้ในการจัดกลุ่มของข้อมูลในตารางผลลัพธ์ โดยยึดเอาฟิลด์ใดฟิลด์หนึ่งเป็นหลักในการจัดกลุ่ม (แต่ก็อาจจะจัดกลุ่มโดยอ้างถึงฟิลด์มากกว่า 1 ฟิลด์ก็ได้) ส่วนมากแล้วก็จะใช้งานร่วมกับฟังก์ชันบางฟังก์ชัน เช่น SUM, AVG และเพื่อสร้างความเข้าใจให้กระจ่างชัดยิ่งขึ้น ขอให้ดูที่ตารางในรูปที่ 5(ก) ซึ่งเป็นตารางเก็บใบกำกับสินค้า เมื่อต้องการหายอดรวมของใบกำกับสินค้า โดยแจกแจงตามรหัสผู้ขาย (ฟิลด์ EMP) ก็สามารถใช้อนุประโยค GROUP BY ในคำสั่งดังนี้

```
SELECT emp, SUM(amount) AS
total FROM invoice GROUP
BY emp
```

และจะให้ผลลัพธ์ดังในตารางที่ 5(ข) ให้ทดลองตัดฟังก์ชัน SUM ออกจากคำสั่ง แล้วทดลองรันดู ให้สังเกตว่าผลลัพธ์ที่ได้แตกต่าง ๆ ไปอย่างไรบ้าง

- HAVING เป็นอนุประโยคที่ใช้ร่วมกับ GROUP BY โดยมีหน้าที่ในการเลือกเอาเฉพาะข้อมูลของกลุ่มที่เข้าในเงื่อนไขที่กำหนดต่อท้ายอนุประโยค HAVING เท่านั้น ไปปรากฏในตารางผลลัพธ์ ซึ่งเมื่อพิจารณาแล้ว HAVING จึงเป็นอนุประโยคที่ขึ้นอยู่กับอนุประโยค GROUP BY นั่นคือจะสามารถใช้ HAVING ได้ก็ต่อเมื่อได้มีระบุอนุประโยค GROUP BY ไว้ในคำสั่งแล้วเท่านั้น (และมีลำดับการจัดวางอยู่ต่อจาก GROUP BY) เงื่อนไขที่กำหนดในอนุประโยค HAVING ส่วนมากจะเป็นการเปรียบระหว่างค่าของฟังก์ชันกับค่าคงที่ใด ๆ ซึ่งได้แสดงตัวอย่างไว้ในรูปที่ 6 (จากตัวอย่างข้อมูลในอนุประโยค GROUP BY)

- ORDER BY เป็นอนุประโยคที่จะเลือกใช้ก็ได้ (หรือไม่ใช้ก็ได้) และถูกจัดวางเป็นอนุประโยคที่อยู่ท้ายสุดของคำสั่ง โดยใช้ในการจัดเรียงลำดับของข้อมูลในตารางผลลัพธ์ ตามค่าข้อมูลฟิลด์ที่ระบุต่อท้ายอนุประโยค ORDER BY (ซึ่งระบุให้มีการเรียงลำดับได้มากกว่าหนึ่งฟิลด์) และยังสามารถกำหนดทิศทางของการเรียงลำดับได้โดยคีย์เวิร์ด ASC ซึ่งเป็นการเรียงลำดับจากค่าข้อมูลน้อยไปหามาก และ DESC ซึ่งเป็นการเรียงลำดับจากค่าข้อมูลมากไปหาน้อย ซึ่งผมขอแสดง

INVOICE	CUST	DATE	EMP	AMOUNT
112233	1024	05/05/91	123	1345.50
112234	4319	05/05/91	451	985.44
112235	2055	05/05/91	155	243.75
112236	4319	05/08/91	451	942.44
112237	2055	05/09/91	155	2440.75

5 (ก)

RESULT	EMP	TOTAL
	123	1345.50
	451	1927.88
	155	2684.50

5(ข)

RESULT	EMP	TOTAL
	451	1927.88
	155	2684.50

5(ค)

RESULT	EMP	COUNT	TOTAL
	155	2	2684.50

5(ง)

รูปที่ 5 ข้อมูลตัวอย่าง/ผลลัพธ์จากการสอบถาม

```
: SELECT emp, SUM(amount) AS total FROM
  invoice;
  GROUP BY emp HAVING COUNT(*) > 1 (จะ
  ปรากฏผลลัพธ์ดังในตารางที่ 5.ค)
: SELECT emp, COUNT(*), SUM(amount) AS
  total FROM invoice;
  GROUP BY emp HAVING SUM(amount) >
  2000.00 (จะปรากฏผลลัพธ์ดังในตารางที่ 5.ง)
```

รูปที่ 6

ตัวอย่างการใช้งานดังนี้คือ ให้แสดงผลข้อมูลในแฟ้ม CUSTOMER.DBF โดยเรียงลำดับตามชื่อเมืองจากน้อยไปหามาก และให้เรียงลำดับยอดสั่งซื้อ/ปีจากมากไปหาน้อย ซึ่งเขียนเป็นคำสั่ง SELECT ได้ดังรูปที่ 7 ครับ

```
: SELECT customer.cno, customer.company,
customer.city, customer.state ;
FROM customer ;
ORDER BY customer. city ASC,
customer.ytdpurch DESC
```

รูปที่ 7 ตัวอย่างคำสั่งที่มีอนุประโยค ORDER BY

ส่วนเพิ่มเติม

นอกจากอนุประโยคที่ได้กล่าวมาแล้วในข้างต้นนั้น ในคำสั่ง SELECT ของฟอกซ์โปรยังประกอบด้วยอนุประโยคอื่น ๆ อีก ซึ่งเป็นอนุประโยคซึ่งใช้ในการระบุปลายทางที่ใช้จัดเก็บข้อมูลผลลัพธ์ โดยในฟอกซ์โปรมีการระบุปลายทางของผลลัพธ์ได้สองแบบคือ

1. **INTO <destination>** ในคำสั่ง SELECT ของฟอกซ์โปร จะให้ค่าความสำคัญของการระบุปลายทางของการจัดเก็บผลลัพธ์ในแบบนี้ มีความสำคัญเป็นลำดับแรก โดยถ้าหากมีการระบุอนุประโยค INTO ร่วมกับอนุประโยค TO ในคำสั่งเดียวกัน ฟอกซ์โปรจะดำเนินการตามวิธีการของอนุประโยค INTO โดยไม่สนใจอนุประโยค TO และถ้าหากไม่มีการระบุอนุประโยค INTO ไว้ในคำสั่ง ฟอกซ์โปรก็จะแสดงผลลัพธ์ออกทางจอภาพ หรือทางอาจจะเปลี่ยนทิศทางของผลลัพธ์โดยการระบุอนุประโยค TO ก็ได้ สำหรับชนิดของปลายทางที่ระบุใน <destination> ของอนุประโยค INTO ประกอบไปด้วยทางเลือกดังต่อไปนี้

- **ARRAY <ชื่อของอะเรย์>** เป็นการระบุว่าต้องการเก็บผลลัพธ์ของการสอบถามเอาไว้ในตัวแปรแบบอะเรย์ ซึ่งได้ระบุชื่อของตัวแปรต่อท้าย

- **CURSOR <ชื่อแฟ้มข้อมูลชั่วคราว>** เป็นการเก็บผลลัพธ์เอาไว้ในรูปแบบของแฟ้มข้อมูลแบบชั่วคราว โดยแฟ้มข้อมูลแบบนี้จะมีคุณลักษณะเหมือนกับแฟ้มข้อมูลของฟอกซ์โปรทุกประการ เพียงแต่ว่าแฟ้มข้อมูลที่กำหนดเป็น CURSOR นี้จะถูกลบทันทีเมื่อทำการปิดแฟ้มข้อมูล

- **DBF <ชื่อแฟ้มข้อมูล> : TABLE <ชื่อแฟ้มข้อมูล>** เป็นการเก็บผลลัพธ์เอาไว้ในรูปแบบของแฟ้มข้อมูล และถ้าหากไม่ได้กำหนดส่วนขยายของชื่อแฟ้มข้อมูล ฟอกซ์โปรก็สร้างแฟ้มข้อมูลที่มีส่วนขยาย .DBF

2. **TO** โดยปกติแล้วถ้าหากคุณไม่ได้ใช้อนุประโยค INTO ในการระบุปลายทางของผลลัพธ์ ฟอกซ์โปรก็จะแสดงผลลัพธ์ออกทางจอภาพ แต่ถ้าหากคุณต้องการจะเปลี่ยนทิศทางของผลลัพธ์ไปยังอุปกรณ์อื่น ก็สามารถทำได้ใช้อนุประโยค TO โดยใช้ TO FILE <ชื่อไฟล์> เพื่อบอกให้ฟอกซ์โปรเก็บผลลัพธ์ลงในเท็กซ์ไฟล์ และถ้าหากต้องการให้ผลลัพธ์ที่ได้จากการสอบถามไปเพิ่มต่อท้ายในไฟล์ที่มีอยู่แล้ว ก็ให้ระบุคีย์เวิร์ด ADDITIVE ต่อท้ายชื่อไฟล์ด้วย และถ้าต้องการให้ผลลัพธ์ที่ได้พิมพ์ออกยังเครื่องพิมพ์ก็ให้ใช้อนุประโยค TO PRINTER นอกจากนี้แล้วในอนุประโยค TO คุณอาจจะใช้อนุประโยคต่อไปนี้ร่วมด้วยก็ได้

: **NOCONSOLE** ใช้เพื่อบอกว่าไม่ให้มีการแสดงผล ข้อมูลผลลัพธ์ออกทางจอภาพ

: **PLAIN** ใช้เพื่อบอกว่าไม่ให้มีการแสดงชื่อฟิลด์ในการแสดงผลข้อมูลผลลัพธ์

: **NOWAIT** ใช้เพื่อบอกให้ฟอกซ์โปรทำการแสดงผลข้อมูลผลลัพธ์โดยไม่ต้องหยุดรอ เพราะปกติแล้วในการแสดงผลข้อมูลผลลัพธ์ ถ้าหากผลลัพธ์มีรายการมากกว่าที่จะแสดงได้หมดในหนึ่งหน้าจอภาพ ฟอกซ์โปรจะหยุดรอให้ท่านกดคีย์ใด ๆ เพื่อดูข้อมูลในหน้าถัดไป สำหรับในแง่ของการใช้งานแล้ว อนุประโยค NOCONSOLE/PLAIN/NOWAIT จะใช้งานร่วมกับการระบุปลายทางข้อมูลโดยการใช้อนุประโยค TO และถ้าหากมีการระบุปลายทางของผลลัพธ์โดยใช้อนุประโยค INTO แล้ว อนุประโยคเหล่านี้จะไม่มีผลต่อการทำงานของคำสั่ง

การสร้างเงื่อนไข

ถือได้ว่าเป็นหัวใจเลยทีเดียวครับสำหรับการสอบถามข้อมูลโดยการใช้คำสั่ง SELECT เพราะว่าท่านจะได้รับข้อมูลผลลัพธ์ตามที่ใจต้องการหรือไม่ก็ขึ้นอยู่กับการตั้งเงื่อนไขนี่แหละครับ ดังนั้นผมจึงได้รวบรวมรายละเอียด

ประเภท	ตัวกระทำการ
การเปรียบเทียบ	=, = =, <>, !=, #, >, >=, <, <=, LIKE
Sub Query	ALL/SOME/ANY, EXISTS
ตรรก	AND, OR, NOT
เซ็ท	IN, BETWEEN

รูปที่ 8 ตัวกระทำการในการกำหนดเงื่อนไข

รูปแบบการใช้

: <ชื่อฟิลด์__1> <ตัวกระทำการเปรียบเทียบ> <ชื่อฟิลด์__2> (ใช้ระบุ Join Condition)

: <ชื่อฟิลด์__1> <ตัวกระทำการเปรียบเทียบ> <นิพจน์>

ตัวอย่าง

```
: SELECT * ;
  FROM customer ;
  WHERE ytdpurch > 1500
```

รูปที่ 9

รูปแบบการใช้

: <ชื่อฟิลด์> [NOT] BETWEEN <ค่าเริ่มต้นขอบเขต> AND <ค่าสิ้นสุดขอบเขต>

ตัวอย่าง

: ต้องการสอบถามข้อมูลในแฟ้ม CUSTOMER.DBF เพื่อหาลูกค้าที่มีอัตราภาษีอยู่ระหว่าง 5.50% ถึง 6.0% สามารถเขียนเป็นคำสั่งได้ดังนี้

```
: SELECT customer.cno, customer.company ;
  FROM   customer ;
  WHERE  customer.taxrate  BETWEEN  5.5
         AND 6.0
```

รูปที่ 10

เกี่ยวกับการตั้งเงื่อนไขเพื่อใช้ระบุถึงลักษณะของผลลัพธ์ที่ต้องการอย่างค่อนข้างละเอียด โดยเราจะเขียนเงื่อนไขต่าง ๆ ที่ตั้งขึ้นขึ้นมาที่อนุประโยค WHERE ก่อนอื่นผมอยากให้ดูตารางรูปที่ 8 ซึ่งผมได้รวบรวมตัวกระทำการในการสร้างเงื่อนไขเอาไว้ทั้งหมด แต่ในตอนนี้ผมจะยังไม่กล่าวถึงตัวกระทำการที่ใช้ร่วมกับ Sub query เพราะเรายังไม่ได้กล่าวถึงเรื่องนี้กันเลย ดังนั้นในตอนนี้เราก็จะมาศึกษาพฤติกรรมของตัวกระทำการทั่ว ๆ ไปกันก่อนครับ

- ตัวกระทำการเปรียบเทียบ ถือว่าเป็นตัวกระทำการหลักที่ใช้บ่อยมากในการสร้างเงื่อนไข และยังมีส่วนเข้าไปพัวพันกับตัวกระทำการประเภทอื่น ๆ ด้วย การทำงานของตัวกระทำการแบบนี้จะเป็นการเปรียบเทียบระหว่างค่าข้อมูลสองค่า ซึ่งอาจจะเป็นการเปรียบเทียบระหว่าง ฟิลด์กับฟิลด์, ฟิลด์กับค่าคงที่/นิพจน์ดังตัวอย่างในรูปที่ 9

- BETWEEN เป็นการหารายการข้อมูลที่มีค่าข้อมูลในฟิลด์ที่กำหนดอยู่ในย่านที่ระบุเอาไว้ ดังรูปที่ 10

- IN เป็นตัวกระทำการที่ใช้ในการทดสอบค่าของฟิลด์กับค่าของนิพจน์ที่อยู่ในรูปแบบของเซต โดยถ้าหากข้อมูลในฟิลด์ที่ทำการทดสอบมีค่าอยู่ในเซตที่ระบุเอาไว้ ข้อมูลรายการนั้นก็จะถูกเลือกไปเป็นผลลัพธ์ ดังรูปที่ 11

- LIKE จัดว่าเป็นตัวกระทำการที่เฉียบขาดมาก ในการเปรียบเทียบค่าระหว่างฟิลด์ในแฟ้มข้อมูลกับนิพจน์หรือค่าคงที่ใด ๆ ที่ผมบอกว่าเฉียบขาดก็เนื่องมาจากว่า เราสามารถใช้ Wildcard เข้าช่วยในการเปรียบเทียบได้ โดย Wildcard ที่ใช้คือ "__" (ตัวอักษรขีดเส้นใต้) หรือ "%" ให้ลองสังเกตการใช้คำสั่งต่อไปนี้ และเมื่อทดลองรันคำสั่งทั้งสองดู จะเห็นว่าให้ผลลัพธ์อย่างเดียวกัน นั่นคือ ต้องการรายการข้อมูลที่มีคำว่า "Computer" ปรากฏอยู่ในชื่อบริษัท แต่ว่าในแบบแรกนั้นใช้ง่ายกว่า และให้ทดลองรันคำสั่งที่ 3 เพื่อทำความเข้าใจกับการใช้ "__" ดังรูปที่ 12

รูปแบบการใช้
: <ชื่อฟิลด์> [NOT] IN <กลุ่มของข้อมูลแบบเซต>
ตัวอย่าง
: ต้องการหารายชื่อบริษัทลูกค้าที่มีรหัสไปรษณีย์อยู่ในกลุ่ม ซึ่งสามารถเขียนเป็นคำสั่งได้ดังนี้
: SELECT customer.cno, customer.company ;
FROM customer ;
WHERE customer.zip IN ("43411", "43506", "43667")

รูปที่ 11

```
: SELECT company ;
  FROM   customer ;
  WHERE  company LIKE "%Computer%"
: SELECT company ;
  FROM   customer ;
  WHERE  AT("Computer",company) > 0
: SELECT company ;
  FROM   customer ;
  WHERE  company LIKE "___x%"
```

รูปที่ 12

- สำหรับตัวกระทำการ ALL/SOME/ANY และ EXISTS ผมจะนำมากล่าวในคราวต่อไป ซึ่งจะเป็นกล่าวถึงเรื่องการสร้างคำถามย่อย (Sub query) ทั้งนี้เพราะเกี่ยวเนื่องกันโดยตรง

เชื่อมโยงเงื่อนไข

จากตารางรูปที่ 8 จะเห็นได้ว่ามีตัวกระทำการทางตรรก (Logical Operator) ประกอบอยู่ด้วย และท่านอาจจะเริ่มสงสัยแล้วก็ได้ว่า ตัวกระทำการทางตรรกเหล่านี้ จะเอามาใช้งานในกรณีใดและอย่างไร เอาล่ะครับผมจะให้คำตอบเอาไว้ก่อนเลยว่า ตัวกระทำการทางตรรกเหล่านี้ ใช้ในการเชื่อมโยงระหว่างประโยคเงื่อนไข ซึ่งใช้ในกรณีที่เรามีเงื่อนไขมากกว่าหนึ่งเงื่อนไข ก็จะใช้ตัวกระทำการ AND หรือ OR เพื่อเชื่อมโยงเงื่อนไขเข้าด้วยกัน โดยถ้าหากใช้ AND ในการเชื่อม ก็จะมีความหมายเป็น <เงื่อนไข__1> และ <เงื่อนไข__2> นั่นคือ ข้อมูลที่จะถูกเลือกเข้ามาในตารางผลลัพธ์ จะต้องอยู่ในเงื่อนไขทั้งสอง แต่ถ้าหากใช้ตัวกระทำการ OR ก็จะมีความหมายเป็นการเลือกเพียงอย่างใดอย่างหนึ่ง (หรือทั้งสองอย่างก็ได้) นอกจากนี้ยังมีตัวกระทำการ NOT ซึ่งมีบทบาทมากเลย เพราะว่าจะใช้งานร่วมได้กับตัวกระทำการแทบทุกตัว โดยถ้าหากมีตัวกระทำการ NOT ร่วมอยู่ด้วย เงื่อนไขนั้นก็จะมีลักษณะกลับความหมายไปทันที ถ้าหากท่านอยากเห็นจริงเห็นจัง ก็ให้ทดสอบโดยการใส่ตัวกระทำการ NOT ในตัวอย่างของตัวกระทำการ IN, BETWEEN, LIKE (โดยใส่ NOT ไว้ข้างหน้าตัวกระทำการเหล่านี้) แล้วลองสังเกตดูผลลัพธ์ว่าแตกต่างไปจากเดิมอย่างไร

มีอีกตอน

ในตอนแรก ผมตั้งใจว่าจะพยายามเขียนบทความตอนนี้ให้เป็นตอนจบ แต่พอเขียนเข้าจริง ๆ ก็จบยังไม่ได้ เพราะว่าเนื้อหาที่จะกล่าวถึงยังไม่หมด และถ้าจะเอาให้จบภายในตอนนี้ บทความก็อาจจะห้วนและขาดรสชาดไปเยอะทีเดียว ก็เลยตัดสินใจว่า จะมีต่ออีกตอนหนึ่ง เพื่อชี้แจงในเรื่องที่ยังคั่งค้างอยู่คือ การสร้าง Sub query, การใช้ฟังก์ชัน, รายละเอียดเรื่องของการ Join และเรื่องปลีกย่อยของการแก้ปัญหาในเรื่องที่ฟอกซ์โปร ไม่สามารถตอบสนองได้อย่าง Full RDBMS (ปัญหาของ Null values) ซึ่งผมคิดว่าในตอนต่อไปจะเป็นภาคสมบูรณ์ของบทความชุดนี้ ซึ่งไม่ควรพลาดอย่างยิ่ง

แล้วเจอกันใหม่ครับ